מטלת מנחה 11 - מערכות הפעלה 20594

שאלה 2

א. פעולת ה TRAP היא פעולה המתבצעת בעת קריאת מערכת, והיא מעבירה את המעבד ממצב משתמש למצב מיוחס. באמצעותה, ניתן לקרוא לשירותים שמערכת ההפעלה מספקת, כמו קריאה/כתיבה לקבצים, יצירת תהליכים בנים וכו'.
מערכת ההפעלה בודקת את הפעולה שנשלחה לבצע, שולפת את הפרמטרים על מנת לבצע אותה, ומבצעת את ההוראות המתאימות במצב מיוחס.

ב. בפעולת ה write, ראשית מעבירה פונקציית הספרייה את הפרמטרים המתאימים למקומותיהם באמצעות אוגרים במעבד: קוד הפעולה Write, ה File Descriptor של הקובץ אליו נרצה לכתוב, כתובת המידע ב RAM אותו נרצה לכתוב, ואורך המידע בבתים אותו נרצה לכתוב לקובץ.

לאחר מכן מתבצעת קריאת המערכת עצמה. מתבצעת פעולת הTRAP, מערכת ההפעלה מקבלת את הפרמטרים מהאוגרים - סוג ההוראה, ופרמטרים של הוראת הכתיבה, וכותבת את המידע הדרוש בהתאם.

במערכת ההפעלה Linux, לאחר קריאת ה TRAP, בעזרת האוגר המייצג את סוג הפעולה המערכת מבינה כי עליה לעשות פעולה מסוג write. היא כותבת לדיסק הקשיח במקום המתאים את המידע המתאים, ומחזירה את מספר הבתים שנכתבו.

ג. יש שני הבדלים מרכזיים בין write וprintf.
ראשית, ב write נוכל לבחור לאיזה קובץ אנחנו כותבים, ואילו ב printf אין לנו שליטה על הקובץ - ונוכל לכתוב לstdout בלבד.
שנית, פעולת ה write מקבלת buffer של מידע לכתוב לקובץ, ואילו פעולת ה printf מקבלת מחרוזת של פורמט וארגומנטים לשבץ בתוכה (גודל לא-קבוע של מחזורת). המשמעות היא שפעולת ה printf מחשבת את גודל הבתים לכתוב בעצמה, והיא מבצעת זאת בעזרת buffer בגודל קבוע, שנכתב בכל פעם שמתמלא, עד לסיום ההדפסה.

שאלה 3

א. מימוש הסמפור, כולל בדיקה עם תהליכונים, בקובץ semaphore.c שבתיקיית הממ"ן. הקוד הרלוונטי לפתרון:

```
typedef struct
{
    int status;
    int waiting; /* represents the "semaphore queue". 0 for an empty
queue, 1 if the queue is not empty. */
} semaphore;

semaphore s;
sigset_t waitset;

/**
 * Initializes the semaphore with a given
 * @param status (0 or 1) represents the number of threads that are
allowed to be running at the same time
 */
void sem_init(int status)
{
    if (!status && status != 1)
    {
        printf("Error: status must equal 0 or 1, current status: %d",
status);
        exit(1);
    }

    s.status = status;
    s.waiting = 0;

    /* Signal waitset config */
    sigemptyset(&waitset);
    sigaddset(&waitset, SIGUSR1);           /* init waiting list */
    sigprocmask(SIG_BLOCK, &waitset, NULL); /* Line copied from
SignaalsOUSSyncrony example in order to make things work. */
}




/**
 * Syncronously wait for signal SIGUSR1
 */
```

```
void wait_for_signal()
{
    int sig;

    /* waiting happens here */
    if (sigwait(&waitset, &sig) != 0)
    {
        printf("Error when waiting for signal");
        exit(1);
    }
}
/**
 * The down method, as stated in the study guide page 57.
 * This is a syncrounous method that wait for the signal SIGUSER1 if
the status of the semaphore is blocked
 */
void sem_down()
{
    if (s.status == 0) /* if semaphore is blocked, mark as waiting and
*/
    {
        s.waiting = 1;
        wait_for_signal();
    }
    else if (s.status > 0) /* if semaphore is not blocked, block it for
future threads. */
        s.status--;
}
/**
 * The up method. Frees the semaphore, or signals another thread to
wake up if waiting
 */
void sem_up()
{
    if (s.waiting == 1)
    {
        kill(getpid(), SIGUSR1); /* send signal to wake up waiting
thread */
        s.waiting = 0;
    }
    else
        s.status++;
}
```

ב. למיטב הבנתי, לא ניתן לממש כך מספר תהליכים ללא סיבוך משמעותי.
הסיבה היא שעבור יותר מ thread אחד שיחכה ל signal, לא ניתן לשלוט באיזה thread יתפוס את הסיגנל ראשון וימחק אותו מרשימת הסיגנלים המתאימים, והוא אינו בהכרח יהיה הראשון בתור ה threads שמחכים להיכנס לקטע הקריטי.

שאלה 4

א. במודל M:1, כל התהליכונים ברמת המשתמש ממופים לאותו תהליכון גרעין. מבחינת תהליכוני המשתמש, תהליכון הגרעין מבצע את הוראות ה CPU שלהם, והם ניגשים אל ה CPU דרכו.

המודל לא מאפשר להריץ מספר תהליכונים במספר ליבות, שכן כל התהליכונים ממופים לאותו תהליך-גרעין, שמשתמש בליבה אחת. ואכן, כתוב:
"The process can only run one user-level thread at a time because there is only one kernel-level thread associated with the process."

ב. במודל M:1, חסימת אחד התהליכונים תחסום את כל התהליך. אכן, כתוב:
"The entire process is blocked if one user-level thread performs a blocking operation."
במודל M:1, חסימת התהליכון תחסום את כל התהליך משום שהיא תחסום את תהליכון-הגרעין המשויך לתהליכון, ותהליכון-גרעין זה הוא יחיד: לא ניתן להריץ אחד אחר במקומו.

ג. במודל 1:1, כל תהליכון-משתמש ממופה לתהליכון-גרעין אחר, דבר המאפשר להריץ תהליכונים שונים של אותו תהליך על ליבות שונות במעבד בעת ובעונה אחת.

משמעות המושגים היא ברמת הניהול: בעוד תהליכוני-משתמש מנוהלים על ידי התהליך המנהל אותם, תהליכוני-גרעין מנוהלים על ידי הגרעין עצמו, והוא זה שמחליט למי להקצות זמן ביצוע.

שאלה 5

נתייחס לאלגוריתם Peterson כפי שמופיע בעמוד 53 במדריך על מנת להסביר איך הקוד עובד.
נראה כי במקרה הגרוע, תהליך יחכה את הזמן שלקח לתהליך השני להיכנס ולצאת מהקטע הקריטי, ולא יותר. זמן זה הוא זמן סופי, ולכן בפרט נוכיח את הטענה.

אכן, נתייחס לתהליך 0 במקרה הגרוע ביותר.
במקרה הגרוע ביותר, תהליך 1 מעוניין להיכנס לקטע הקריטי גם הוא, ולכן $interested[1] = true$.
כאשר תהליך 0 מתכוון להיכנס לקטע הקריטי, מתבצעת השורה $turn = 0$ ולכן התנאי בלולאת ה while בשורה הבאה מתמלא ותהליך 0 נכנס להמתנה עסוקה (busy wait).

כאשר מתבצע context switch לתהליך 1, התנאי בלולאת ה while בהכרח לא מתקיים (כי turn=0), ולכן מתבצע הקטע הקריטי. לאחר מכן מבוצע $interested[1] = false$, ותנאי הwhile של תהליך 0 כבר לא מתקיים.

לכן, כשיתבצע context switch לתהליך 0, גם תהליך 0 ייכנס לקטע הקריטי.
זמן ההמתנה, במקרה הגרוע ביותר הזה, הינו הזמן שלקח לתהליך 1 להיכנס ולעזוב את הקטע הקריטי.

ביצועי context switch נוספים טרם יציאת תהליך 1 מהקטע הקריטי הם חסרי משמעות - משום שהם נכללים בזמן שלקח לתהליך 1 לצאת מהקטע הקריטי.